CSE4939W and CSE 4940 CSE Design Laboratory I & II

DUE DATES: Sprint #1 (First draft – 9/30, Revised – 10/7)
Sprint #2 (First draft – 11/4, Revised – 11/11)
Sprint #3 (First draft – 12/11, Revised – 01/27/2016)

You will build your project following the agile software development process which is documented on the web page. Specifically:

- An overview of the agile and scrum process can be found at – please see: scrumreferencecard.com
- The web site has number of documents of note.
    1. There is a six page PDF that explains the agile process including sprints and the overall scrum – please see: scrumreferencecard.com/ScrumReferenceCard.pdf
    2. There is also a PDF with check list for the scrum master to use during the review process of the sprint – please see: scrumreferencecard.com//ScrumMasterChecklist.pdf

At the core of the agile process, the project is built over several iterative incremental cycles called as Sprints. With each sprint, you will update and deliver this document to reflect the changes and additions to the project. The template has the following sections, with guidelines following that explain the contents of each section in detail.

Section A. Sprint Backlog – these are the list of features/capabilities that are the focus on the current sprint – they are the intended deliverables for the project.

Section B: User Stories/Use Cases – this is utilized to define the user stories (scenarios) and use case (UML use-case diagrams with explanation) for the project.

Section C: User Based Specification/Interfaces – this allows the definition of user interfaces (UIs) that are essentially mockups of the various functions that support the features from the sprint and area used to realize in part the user stories/use cases. Note that over time, each of these mockups will transition to actual screen shots – so provide the mock and a description of its function. Then, this section of the sprint document can evolve to the user manual at the end of the academic year.

Section D: Detailed Design – this models the system using UML diagrams, software design patterns, entity relationship diagrams, etc.

Section E: Test plans – this allows you to track the test cases and results for the implementation associated with the current sprint.

Section F: Product Backlog Items (PBI) – while defined initial in the high-level project specification, this should be migrated into the sprint document and updated for each sprint to be consistent with the Sprint backlog (which is always a subset of PBI).

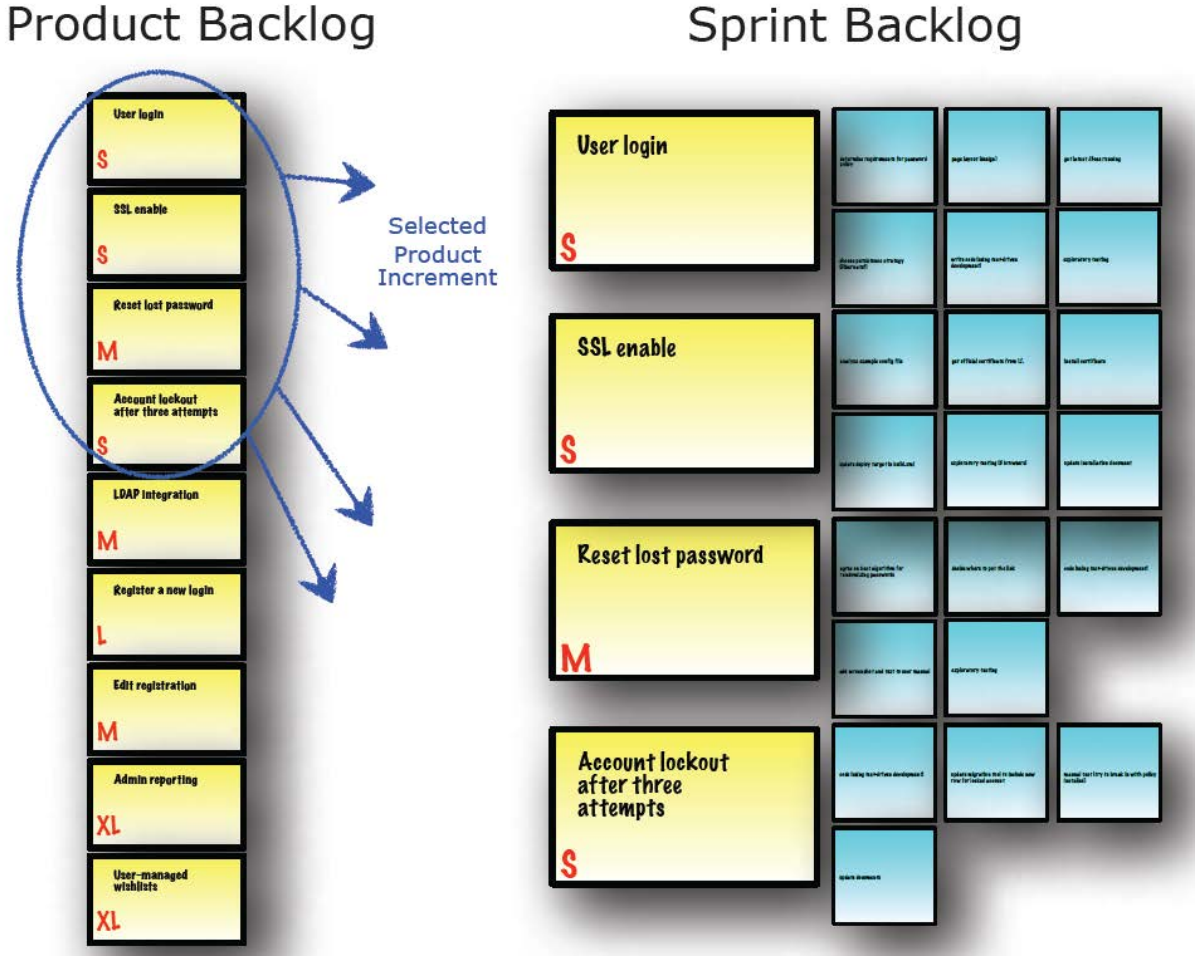
For examples of the sprint document for Teams D and G see:

- http://www.engr.uconn.edu/~steve/Cse4939W/TeamDInitSprint.docx
- http://www.engr.uconn.edu/~steve/Cse4939W/TeamGInitSprint.docx

For the initial Sprint due on September 30, please provide only Sections A, B, C, and F. All subsequent sprints will require all Sections A to F.

Section A: Sprint Backlog

The sprint backlog is a list of tasks identified by the scrum team to be completed during the scrum sprint. You must consult the list of Product Backlog Items (PBIs) identified from your specification document to come up with the sprint backlog. Also, note that this list may contain new features as well as fixing of bugs that were revealed during the previous sprints. Below is a sample of the PBI and its association with the sprint backlog where the size/complexity of each item is labeled with: where S=Small, M=Medium, L=Large, and XL= extra L.



Notice that the sprint backlog is a subset of PBI and for each item included, a corresponding set of sprint items are defined (blue boxes in figure). The information in both the PBI and sprint backlog can be conceptualized in spreadsheet form. Below are samples of each in spreadsheet form (see slides 34 to 37 of www.engr.uconn.edu/~steve/Cse2102/finagile.pptx.

For each listed task, you should also identify: the primary and secondary member of each task (initials), the name of the task, and the start date and end date for each task. This is slightly differen than what is shown in the spreadsheet below, where the number of hours have been used to estimate effort across the four weeks of the sprint. The two initial sprints for Teams A, D and G on the course web page contain a slightly revised version of this spreadsheet – please follow that format. Notice in the spreadsheet that each PBI is identified (e.g., User's Guide) and the tasks are then listed for each one.

Sprint Backlog

		Days Left in Sprint	15	13	10	8	
Who	Description		7/22/2002	7/24/2002	7/26/2002	7/31/2002	
		Total Estimated Hours:	**554**	**458**	**362**	**270**	**0**
-	**User's Guide**		-	-	-	-	-
SM	Start on Study Variable chapter first draft		16	16	16	16	
SM	Import chapter first draft		40	24	6	6	
SM	Export chapter first draft		24	24	24	6	
	Misc. Small Bugs						
JM	Fix connection leak		40				
JM	Delete queries		8	8			
JM	Delete analysis		8	8			
TG	Fix tear-off messaging bug		8	8			
JM	View pedigree for kindred column in a result set		2	2	2	2	
AM	Derived kindred validation		8				
	Environment						
TG	Install CVS		16	16			
TBD	Move code into CVS		40	40	40	40	
TBD	Move to JDK 1.4		8	8	8	8	
	Database						
KH	Killing Oracle sessions		8	8	8	8	
KH	Finish 2.206 database patch		8	2			
KH	Make a 2.207 database patch		8	8	8	8	
KH	Figure out why 461 indexes are created		4				

Section B: User Stories/Use Cases

A user story is a short description of something that your customer will do when they come to your website or use your application/software, focused on the value or result they get from doing this thing. They are written from the point of view of a person using your website or application, and written in the language that your customers would use. For example, as a Facebook user I want to set different privacy levels for status updates and photos. This will allow me to control who sees what I share.

A use case is a description of a set of interactions between a system and one or more actors (where 'actor' can be people, or other systems: for example, both online shoppers and credit card databases can be actors). Sometimes use cases are written to flesh out user stories in the agile development process. Use cases generally include this information:

- Title of the use case
- Rationale/description/goal
- Actors/users
- Preconditions (the things that must have already happened in the system)
- Standard path or Main success scenario (what will usually happen, described as a series of steps)
- Alternate paths or Extensions (variations on the above/edge cases)
- Post conditions (what the system will have done by the end of the steps).
- Any other use cases that the use case may include. The included use case may not be limited to the present increment.

For the first three sprints, include 5-7 use cases, in one or more diagrams. You can alter and/or add to these diagrams in the spring, if need be, in the event new actors are discovered or new PBI entries cause reassessment.

You can seek help from the following resources, which discuss how to write use cases for agile projects:

1. http://breathingtech.com/2009/writing-use-cases-for-agile-scrum-projects/
2. http://blog.casecomplete.com/post/Agile-Use-Cases-in-Four-Steps

As shown in section B of initial sprints for Teams A, D and G on the course web page, each figure is identified with a number and labeled with a caption. The figure should be cited in the writeup.

Section C: User-Based Specification/Interfaces

This primarily consists of rapidly prototyped graphical user interfaces (via Visio, html, etc.) that can serve as the basis for a dialog between the users and developers in order to extract user requirements and detail required functionality and interactions between various components of the system from the GUI perspective. From a documentation perspective, the screens should be labeled figures organized into a logical order to demonstrate the user interface capabilities. Each screen shot should have an accompanying paragraph description to explain its purpose and usage.

As previously mentioned, the intent is that these mockups will evolve to actual screen shots for the working application. As this transition occurs, the mockups should be replaced by screen shots in successive sprints. As a result, this section of the sprint document will evolve over time to the user manual that is required in the formal submittal in the Spring semester. Please see the initial sprints for Teams A, D and G on the course web page for examples of these mockups and write-ups.

As shown in section C of initial sprints for Teams A, D and G on the course web page, each figure is identified with a number and labeled with a caption. The figure should be cited in the writeup.

Section C: Detailed Design

The detailed design for the project is the third part of the Sprint document that is delivered with each increment and represents a series of software design artifacts, models, and techniques, accompanied by associated written documentation, that fully describes the software design as a lead in to the development and testing stages. The project design should include the following parts:

Software Design Patterns (SDPs): Software design patterns typically show *relationships* and *interactions* between classes or objects at a conceptual level, for domain independent structure and behavior. A pattern provides a body of knowledge on a particular structure thereby communicating insight on a portion or component of a solution; the idea is that we can leverage patterns that are generalized from prior solutions and experience to build solutions more effectively**.** Each project will need to identify, define, explain, and document a set of one or more SDPs including: High-level Patterns such as Model View Controller or MVC with Observer; Creational Patterns such as Abstract Factory, Builder, etc.; Structural Patterns such as Adaptors, Proxy, etc; Behavioral Patterns such as Interpreter, Mediator, etc.; and Concurrency Patterns such as Event-Base Asynchronous, Scheduler, etc.

1. **Detailed Design:** This includes software and database design. For software design, you are to employ a wide range of **UML** diagrams as appropriate for your particular project and its domain requirements: Structural UML diagrams include: Class, Component, Object, Profile, Composite Structure, Deployment, and Package. Behavioral UML diagrams include: Activity, State Machine, Sequence, Communication, Interaction, and Timing. For database design, you are required to construct an entity-relationship diagram (**ERD**) that for the conceptual structure of the data and its interrelations to be used to eventually generate a relational database schema. Note that not all projects may have all of these diagrams – but each project should be able to select at least enough different software and database diagrams so that each team member is responsible for one aspect of the software design as realized by a respective diagram. Some details on different diagrams are:
    - An Entity-Relationship (ER) diagram for shows the conceptual structure of the data and its interrelations (see Ghezzi, section 5.6.1, or a database management textbook). You must use the ER specification capabilities of a UML tool (Eclipse plug in or Visio) in constructing your diagram. . Make sure that you describe the attributes of each entity including types and acceptable ranges. For relationships, make sure you identify their directions and their cardinalities.
    - UML Sequence Diagrams to show the sequential flow between your various system objects and components.
    - UML Statechart Diagrams to show the to show the general control structure of the system and identify each state and the transitions between states.
    - UML Class/Object Diagrams that include both public, private, and protected methods, and private and protected attributes. Organize your classes into packages as appropriate, and use the various UML references and relationships to indicate which classes and/or packages collaborate with one another. You may divide the packages and classes between team members, with each team member responsible for a set of packages and classes if this aspect of your design is significant in size.

- UML Activity Diagrams (ADs) to show structural components and flow between the various states that support both forms and joins to indicate actions that can occur in parallel.

Note that a particular sprint may not employ any new design patterns or UML diagrams over the previous sprints. This design will continue to be updated in the remainder of the semester and during the second semester (CSE4940) as the direction and approach changes based on problems identified, new requirements, etc.

- Each of the first three sprints must have at least 5 of the selected diagrams.
- Each team member is responsible for one of these diagrams.
- Each diagram must be accompanied with a textual explanation. The rule of thumb is ½ page of description per ½ page of diagram.
- You can modify or add to these diagrams in the spring based on the reassessment of the PBI.
- For UML, there are a myriad of sources online including: http://www.uml.org/. An image search on your favorite search engine with "UML Diagrams" will show lots of samples.

As shown in section D of initial sprints for Teams A, D and G on the course web page, each figure is identified with a number and labeled with a caption. The figure should be cited in the writeup.

Section E: Test Plans

You will perform different types of possible testing depending on your project and the presentation of your project on SDP day on the last day of the semester. Each increment must be accompanied by a focused testing regime that occurs at varying levels of granularity, spanning white box testing of the code (method level testing) to general performance testing that varies based on the type of application and its specific needs to evaluating software qualities. There will be five testing exercises this semester, corresponding to each increment after the first, with the intent that each team member takes the lead in learning about one of the testing techniques so that the entire team is able to conduct sufficient testing using that technique for each increment. There is a PPT on the web page from the software engineering course on testing, if you need to review the various concepts (http://www.engr.uconn.edu/~steve/Cse230/finchapter6.ppt).

Unit and white-box testing; testing in the small

Module testing or testing in the small using white-box testing is the first approach to take when focused on the functional and correctness of the code, typically emphasizing the testing of public, protected, and private methods of classes. In the Java world, unit testing is supported by many products, such as junit (http://junit.sourceforge.net/) and there are many good tutorials available for its usage: http://www.vogella.com/articles/JUnit/article.html and also http://www.tutorialspoint.com/junit/index.htm. For objective-C, there are many tools such as OCUnit (http://cocoadev.com/wiki/OCUnit) and GHUnit (http://gabriel.github.com/gh-unit/), and there are many tutorials on line for you to review. For C, there is Check for C (http://check.sourceforge.net/), C/C++test http://www.parasoft.com/jsp/products/cpptest.jsp, and googletest for C++ applications. In the Microsoft world, MSTest can be utilized with Visual Studio (http://msdn.microsoft.com/en-us/library/ms182489%28v=vs.80%29.aspx). For mobile applications in general, the appcelerator (Titanium) recommends jsunity (https://github.com/atesgoral/jsunity), s Universal framework for Javascript. Qunit is another Javascript unit testing framework (http://qunitjs.com/). If you are using a gaming framework, it may have its own unit testing support. For example. http://www.prosoxi.com/2011/11/26/unit-testing-with-cocos2d-and-xcode-4/ can be used for unit testing with Cocos2d and Xcode (https://developer.apple.com/xcode/). For Unity3D, one testing source is (http://blogs.unity3d.com/2012/05/08/testing-unity/). For Sencsha touch, see (http://www.sencha.com/blog/ui-testing-a-sencha-app/) . For Django, see: https://docs.djangoproject.com/en/dev/topics/testing/?from=olddocs . and html5 see: the jasmine product (https://github.com/pivotal/jasmine). In summary, each team must choose the testing approach most appropriate for their project.

User Interface Testing

Graphical user interfaces (GUI's) testing is necessary to ensure correctness of the business or program function and logic. For example, only allowing correct inputs into fields, checking formats of dates or phone numbers, making sure minimums are entered for a particular screen or stage of the software before allowed to go forward, etc. Or for games, checking that all of the buttons (on touch screen) and/or options work correctly; you can even check to be sure there is

consistency in placement across multiple screens . For web or mobile apps, checking individual screens, order of screens, minimums need to be able to complete a screen to submit or go to the next screen, are all reasonable approaches. Each team needs to make a decision in regards to the focus of GUI testing that is most relevant for the project. You can even use a scenario based approach that enumerates possibilities that can occur on a given screen that would cause some action at the server or database. But the key focus is on ensuring correct input. As examples, the course web page has three files: TestScenario.pdf (Scenario link) that contains a GUI screen and an elaborate set of tests to be conducted; WebTesting.pdf (Web App link) that contains testing results for web app; and, LicenseTesting.pdf (Java App link) that contains testing results for Java desktop application. The first file eliminates a scenario to test for in a Java desktop application; you can see that the testers (domain users) are attempting to enumerate all of the possible ways that the screen can be used for processing (business logic) and trying cases that should succeed, as well as one that should fail. These results as well as the information on who tested what is carefully tracked. In the other two files, you can see some of the results of the testing, where the end users (tester) takes screen shots and annotates each screen with the appropriate changes and/or corrections. To collect, track, and annotate your testing, you can use a screen capture application (e.g., http://camstudio.org/ or http://www.wisdom-soft.com/products/screenhunter_free.htm).

General Performance Testing

General performance testing has a wide range of possibilities that are chosen based on your application. For example, web and mobile applications often to what is called stress testing where a person is testing a particular application to see at what points the application breaks, to determine the amount of simultaneous users, to assess performance as users increase, etc. One such tool to utilize is jmeter (http://jmeter.apache.org) which can send multiple http requests to test traffic capacity on a server. Stress testing can also be used to try to inundate the web server, application server, or database server. Another product has different levels of web testing for loads, and can record and replay tests (http://www.webperformance.com/). For teams doing interactive multi-player applications games, testing on different loading in terms of user, size of maps, etc., are all relevant to understand essentially under what conditions a system may fail or performance starts to degrade. A third type of testing may be if there is some aspect of your system that requires a high amount of CPU cycles and/or memory to operate – perhaps there is a complex algorithm. In this case, you can put checkpoints (calls to the system clock) before and after a piece of code to record timing based on different loads, and record these time and events in the actual code. Another type of testing can be database related, and there is analog to JUing called DBUnit (http://www.dbunit.org/) to search large repositories. Also – for databases, you have to worry about preventing (at the GUI or web service or API level) users that input SQL commands that could be inadvertently applied to the database to return information – this is called SQL injection testing. There is SQL injection testing for web apps operating in a browse (like Firefox – see https://addons.mozilla.org/en-us/firefox/addon/sql-inject-me/) . The basic idea is to ensure that potential malicious users are not attempting to input SQL commands into a data entry field (e.g., last name, address, etc.) in order to attempt to access the database directly.

Software Quality Attainment

Recall there was a software quality assurance project involving the qualities that are non-function in nature such as: understandability, maintainability, evolvability, and user friendliness. What is interesting about this type of testing is for you and your team to try to ascertain the way that these capabilities can be measured in some way. Is your system each to understand from both the developer and end user perspectives? How can you demonstrate its maintainability? Was there a recent problem that you identified and fixed, and if so, was it easy or hard? Can you make changes easily? Was there an evolution you had to do in the last prototype that wasn't anticipated, and if so, was it difficulty or easy? How user friendly is your system? Did you get any feedback from the black box testing? There are a number of sources on line that you can consider: http://www.softwareqatest.com/index.html and http://www.aptest.com/resources.html. It is your responsibility to explore SW quality attainment and the degree that your final system is able to do so.

Section F: Product Backlog Items (PBI)

The PBI as defined in the high-level project specification, should be migrated into the sprint document and updated for each sprint to be consistent with the Sprint backlog (which is always a subset of PBI). When new features or requirements are discovered they are inserted into the PBI in order for them to be included in a future sprint. When items are completed, that is indicated as well.

You can use a visual representation of the PBI as given below. For each yellow block, you can expand its size and add in a written explanation of each item. See examples in Section F of the initial sprints for Teams D and G on the course web page.